The Mach System

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne

Presented by: Jee Vang
Outline

- Context
- Goals
- History
- Overview of Mach
- Kernel
- System Components
- Process Management
- IPC
- Memory Management
- Programmer Interface
- Summary
Context

The Mach System

- Threads
- RPC
- Layers
Goals

- BSD compatible
- Lightweight kernel
- Support multiprocessors
- Heterogeneous and portable
- Distributed operation over networks
- Efficient management
  - CPU, Memory, Communication
History

- Start with 4.2BSD kernel
- Gradually replace components w/ Mach code
- Inherited Benefits
  - Simple programmer interface
  - Easy portability
  - Extensive library of utilities and apps
  - Ability to combine utilities easily via pipes
- Inherited Drawbacks
  - Bloated kernel
  - No support for multiprocessors
  - Too many fundamental abstractions
Overview of Mach

- Object-oriented
- Message-based
- Transparency through abstraction
- Modular and Portable
  - Objects can reside anywhere since messages are sent to ports
  - Kernel caches the contents of memory objects in local memory
Kernel

“Simple, extensible kernel, concentrating on communication facilities”
System Components

The Key Ingredient

- What makes Mach different?
  - LRPC
    - Only A-stack is shared
    - Everything else remains the same
    - Not much improvement
  - URPC
    - Two threads share memory (communication)
    - Both threads have to constantly poll
    - Kernel is oblivious
  - Mach
    - Blends memory management and IPC features
The Key Ingredient (cont.)

- Memory management
  - Memory objects contain ports that receive IPC messages
  - Allows memory objects to be in remote locations

- IPC
  - When sending messages, memory objects are not copied. Instead, pointers to those memory objects are moved.
  - Less overhead --> Higher efficiency
Process Management

- Tasks and Threads
- C Threads
- CPU Scheduler
- Exception Handling
Process Management
Tasks and Threads

- A task initially has one thread ("process")
- Threads in a task share the task’s resources
- Threads execute independently
- Two states:
  - Running (executing or waiting), otherwise
  - Suspended
- When a task is suspended, all of its threads are also suspended
Process Management
C Threads

- Abstract interface to manage threads
- Major influence of POSIX P Threads standard

- Contains routines to:
  - Create new thread
  - Destroy child thread and return value
  - Wait for a thread
  - Yield control of processor
Process Management
C Threads (cont.)

- Also contains mutex routines:
  - mutex_alloc, mutex_free
  - mutex_lock, mutex_unlock

- As well as condition variable functions:
  - condition_alloc, condition_free
  - condition_wait, condition_signal
Process Management

CPU Scheduler

- Oblivious of tasks. Only threads are scheduled.
- Threads compete for CPU time
- Dynamic priority number [0, 127]
  - Based on CPU usage
- Threads placed into a queue
  - 32 global + 1 local queue per processor
Process Management
CPU Scheduler (cont.)

- No central dispatcher
  - When CPU becomes idle, it picks a queue
  - Local queue has higher priority
- Processor time quantum $\propto \frac{1}{\# \text{ of threads in system}}$
Process Management

Exception Handling

- Exception handler is simply a thread in the task
- Implemented via RPC messages
- Disruptions come in two flavors:
  - Exceptions: Internal. Due to unusual conditions.
- Two granularities of exceptions:
  - Error handling (per-thread)
  - Debuggers (per-task)
- When an exception occurs...

```
Task

wait()

Victim Thread

raise(exception info, thread, task)

Exception Handler

(Clear exception && Resume thread )
|| (Terminate thread )
```
Process Management
Exception Handling (cont.)

- When a hardware interrupt occurs…

- Exception RPC sent to thread that caused exception
- Exception condition gets cleared
- Initiating thread continues executing
- It immediately sees the signal
- Executes its signal-handling code
IPC

- Two components: Messages and Ports
  - Location independent. Highly portable.
- When a task is created, kernel also creates several ports
  - System calls provided to ports
    - Allocate a new port to work on a specified task
    - Deallocate a port. If it has receive right, another port is given the right
    - Get current status of the port
    - Create a backup port
- Security: Senders and receivers have rights
  - Port name and capability (send or receive)
- On each port, only one receiver
  - Receive right can be transferred through a message
- Ports can be used for synchronization purposes
  - For $n$ resources, $n$ messages can be sent to a port
  - Threads needing that resource send receive call to port
  - If a message is received from the port, resource is available
  - Otherwise, retry later
  - After using resource, threads can send a message to the port
  - Keeps track of the number of instances available for that resource
IPC

- Network Message Server (NetMsgServer)
  - Used when receiver port is not on machine
  - User-level capability-based networking daemon
  - Forwards messages between hosts
  - Protocol independent (Portable)

Memory Management

- Memory Objects
  - Used to manage secondary storage
  - Contains ports that receive IPC messages
  - Treated as all other objects in Mach
  - Independent of the kernel
    - Kernel does not assume anything about contents and importance of memory objects
- Mapped into virtual memory
  - The pageout daemon uses FIFO to replace pages as necessary
  - Copy-on-write is used to avoid actually copying the data
Memory Management (cont.)

- **Memory Managers**
  - Pages memory for cheap access
  - Maintains memory consistency
  - Writes "dirty" pages when object is destroyed
  - Mach has its own ("default memory manager")

- **Shared Memory**
  - Reduces overhead --> Fast and efficient IPC
  - Locks and conditions used to protect shared data within a task
  - External memory managers keep shared memory consistent
- How a thread accesses data…
  - `vm_map(mem_object_port, mem_mgr)`
  - Kernel asynchronously executes call on specified port
    - True to form, kernel focuses on communication
    - It does not care whether the call is executed correctly
    - That is the memory manager’s responsibility
  - `memory_manager_init(ctrl_port, name_port)`
Programmer Interface

- System calls
  - Emulation library
    - Contains functions moved from kernel to user space
    - Lives in address space of the program
    - If function not found in library, kernel asks a server
- C Threads provide interface to manipulate threads
- Mach Interface Generator (MIG)
  - Input: Definition of the interface
  - Output: RPC interface code
“Mach is an example of an object-oriented system where the data and the operations that manipulate that data are encapsulated into an abstract Object”